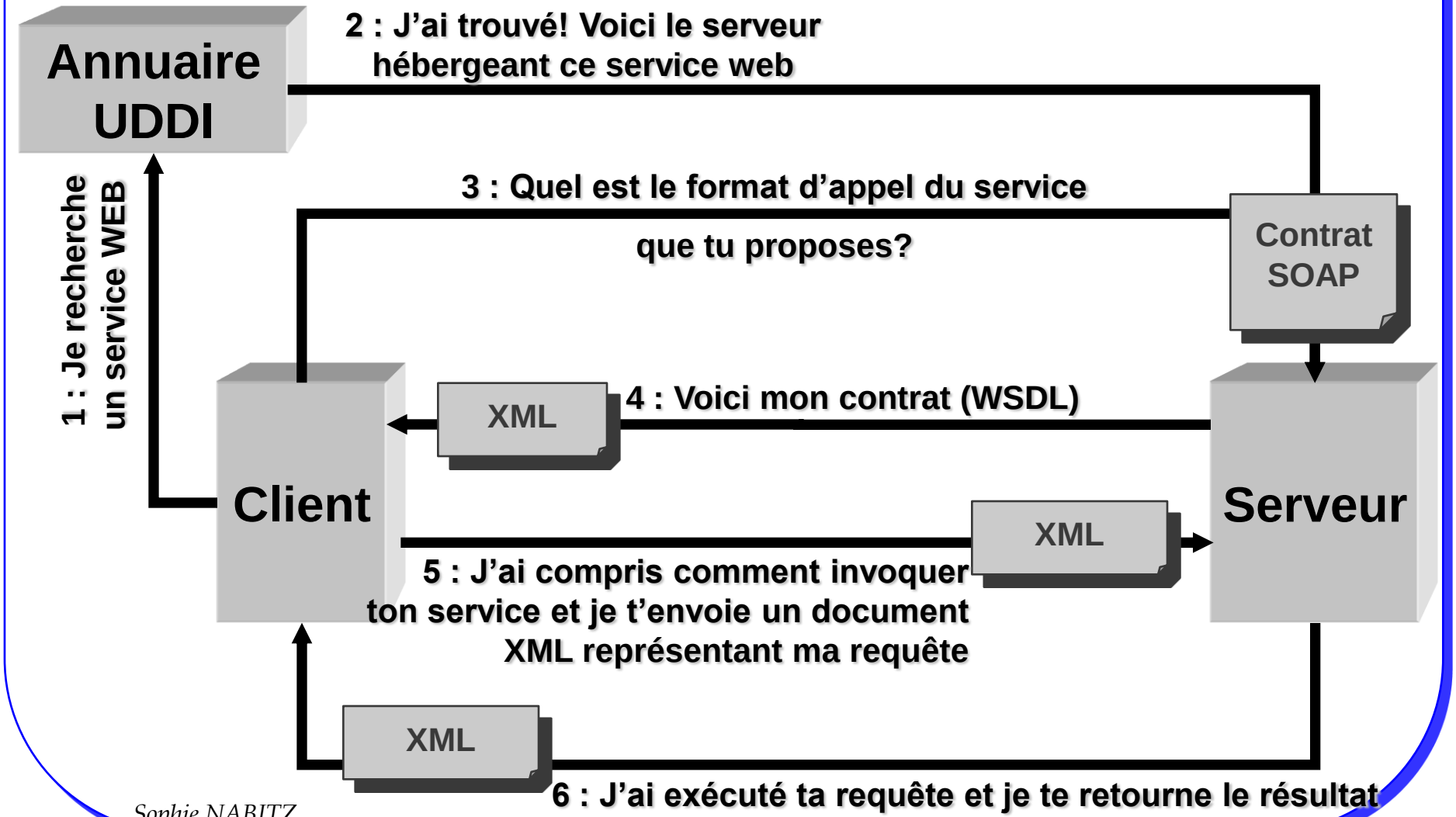
Documents

- La première partie des transparents est une reprise des documents de Jérémy Fierstone disponibles sur l'internet
- La seconde partie des transparents est une reprise des documents de M. Baron disponibles sur developpez.com

La philosophie SOAP

- SOAP codifie simplement une pratique existante
  - Utilisation conjointe de XML et HTTP
- SOAP est un protocole minimal pour appeler des méthodes sur des serveurs, services, composants, objets
  - Ne pas imposer une API ou un runtime
  - Ne pas imposer l'utilisation d'un ORB (CORBA, DCOM, …) ou d'un serveur web particulier (Apache, IIS, …)
  - Ne pas imposer un modèle de programmation
    - Plusieurs modèles peuvent être utilisés conjointement
  - Et "ne pas réinventer une nouvelle technologie"
- SOAP a été construit pour pouvoir être aisément porté sur toutes les plates-formes et les technologies

Cycle de vie d'utilisation

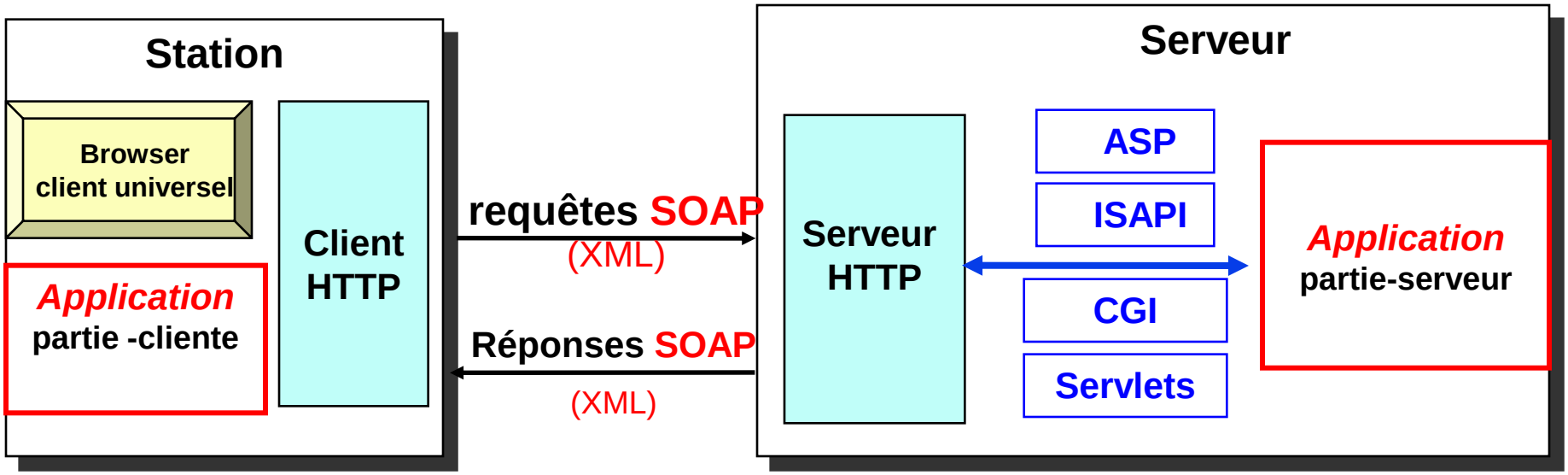


Cycle de vie complet

- Étape 1 : Déploiement du service Web
  - Dépendant de la plate-forme (Apache : WSDD)
- Étape 2 : Enregistrement du service Web
  - WSDL : description du service
  - Référentiels : UDDI
- Étape 3 : Découverte du service Web
- Étape 4 : Invocation du service Web par le client

En résumé

SOAP = HTTP + XML



SOAP sur HTTP

- Utilise le modèle POST Requête/Réponse
- Requête
  - Type MIME : text/xml
  - Champs d'entête supplémentaire de la requête
    SOAPAction : URI
    SOAPAction: "http://electrocommerce.org/abc#MyMessage"
    SOAPAction: "myapp.sdl"
  - Envelope SOAP
- Réponse
  - Status
    - 2xx : le récepteur a correctement reçu, compris et accepté le message inclus
    - 500 (Internal Server Error): le récepteur n'accepte pas le message
  - Envelope SOAP
    - La réponse
    - Le détail des erreurs

Types de message SOAP

- SOAP définit trois types de message
  - Appel (Call) - obligatoire
  - Réponse (Response) - optionnel
  - Erreur (Fault) - optionnel

Appel simple

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml
Content-Length: nnnn
SOAPMethodName: Some-Namespace-URI#GetLastTradePrice

<SOAP:Envelope xmlns:SOAP="urn:schemas-xmlsoap-org:soap.v1”>
   <SOAP:Body>
      <m:GetLastTradePrice
       xmlns:m="Some-Namespace-URI”>
         <symbol>DIS</symbol>
      </m:GetLastTradePrice>
   </SOAP:Body>
</SOAP:Envelope>
```

Réponse

HTTP/1.1 200 OK
Content-Type: text/xml
Content-Length: nnnn

```
<SOAP:Envelope xmlns:SOAP="urn:schemas-xmlsoap-org:soap.v1”>
   <SOAP:Body>
      <m:GetLastTradePriceResponse
       xmlns:m="Some-Namespace-URI”>
         <return>34.5</return>
      </m:GetLastTradePriceResponse>
   </SOAP:Body>
</SOAP:Envelope>
```

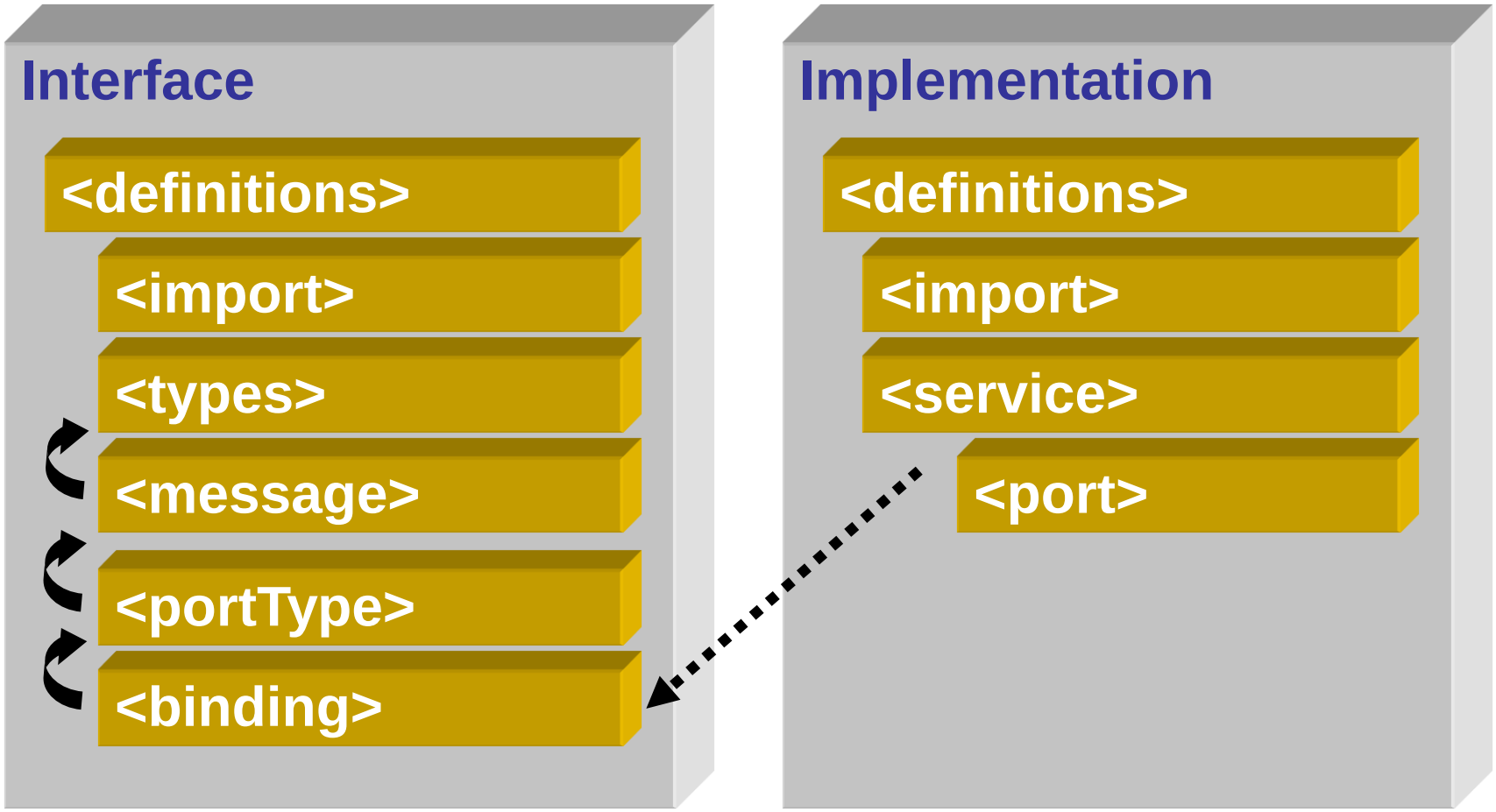
Erreur

```
<SOAP:Envelope xmlns:SOAP="urn:schemas-xmlsoap-org:soap.v1>
    <SOAP:Body>
        <SOAP:Fault>
            <faultcode>200</faultcode>
             <faultstring>
                SOAP Must Understand Error
            </faultstring>
        </SOAP:Fault>
    <SOAP:Body>
</SOAP:Envelope>
```

WSDL

- Spécification (09/2000)
  - proposition de la part de Microsoft, IBM et Ariba
  - TR W3C v1.1 (25/03/2001)
- Objectif
  - Décrire les services comme un ensemble d'opérations et de messages abstraits reliés (bind) à des protocoles et des serveurs réseaux
- Grammaire XML (schéma XML)
  - Modulaire (import d'autres documents WSDL et XSD)
- Séparation entre la partie abstraite et concrète

WSDL


Interface
<definitions>
<import>
<types>
<message>
<portType>
<binding>
Implementation
<definitions>
<import>
<service>
<port>

Éléments d'une définition WSDL

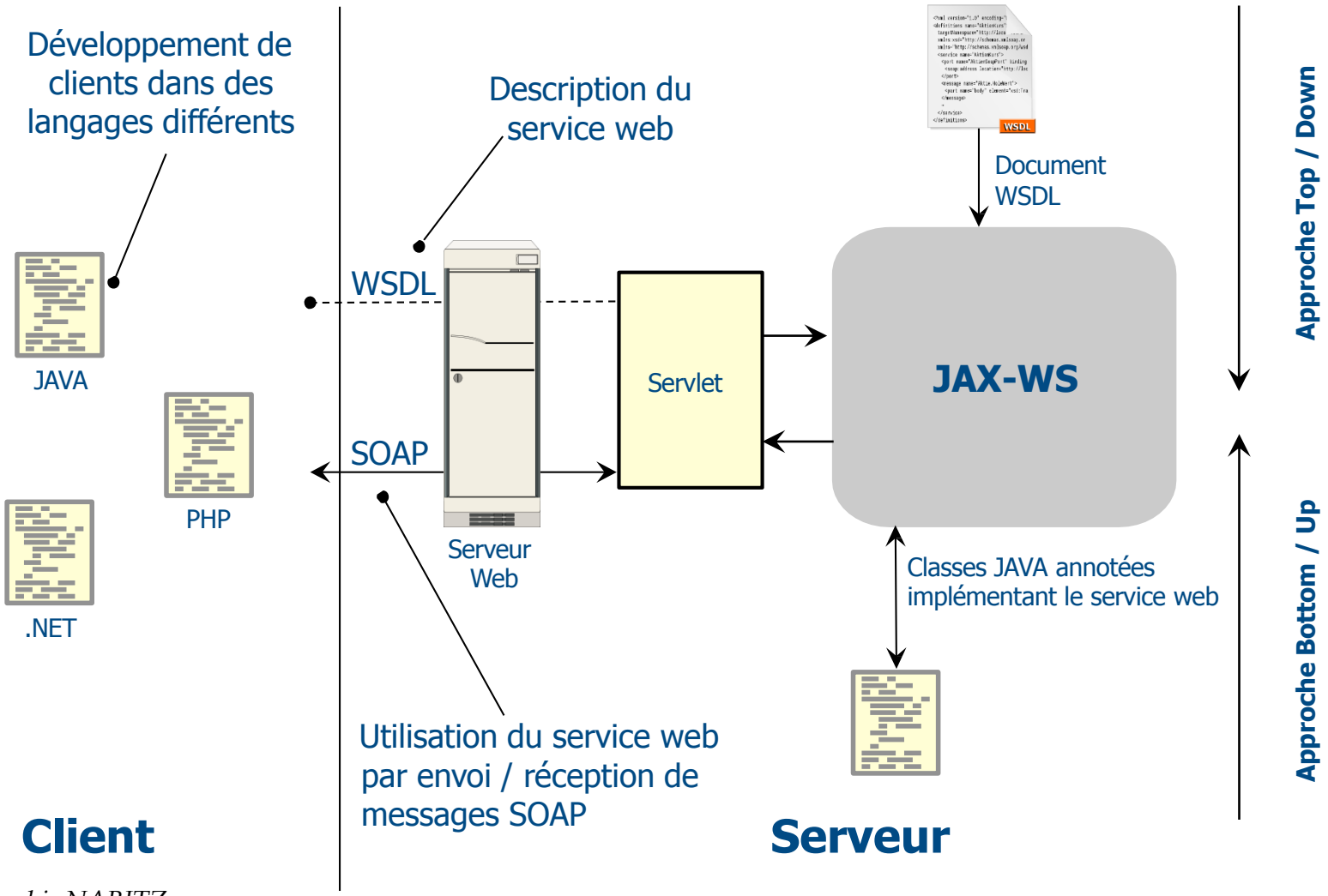
- <definitions> : élément racine du document, il donne le nom du service, déclare les espaces de noms utilisés, …
- <types> : contient les définitions de types (en utilisant un système de typage comme XSD)
- <message> : décrit les noms et types d'un ensemble de champs à transmettre : paramètres d'une invocation, valeur du retour, …
- <porttype> : décrit un ensemble d'opérations
  - chaque opération a zéro ou un message en entrée, zéro ou plusieurs messages de sortie ou de fautes
- <binding>: décrit concrètement comment le service sera implémenté : protocole et format des données
- <service> : définit les adresses permettant d'invoquer le service

Exemple

JAX-WS

- JAX-WS est l'acronyme Java API for XML Web Services
- JAX-WS est à la fois un standard et une implémentation
- L'implémentation JAX-WS est intégrée nativement à la JRE depuis la version 6
- JAX-WS est un sous projet du SA Glassfish
- L'implémentation de référence est fournie par METRO
- Il est possible de développer des services web en dehors d'un serveur d'application en mode autonome

Généralités JAX-WS



Généralités JAX-WS

- Le développement de services web avec JAX-WS est basé sur des POJO (Plain Old Java Object)
- Les fonctionnalités de base pour le développement de services web avec JAX-WS requiert simplement l'utilisation d'annotations Java
- Par conséquent aucun fichier de déploiement n'est requis
- JAX-WS permet d'assurer l'indépendance du protocole (SOAP) et du transport (HTTP)

Développement Serveur

- Deux façons pour développer un service web avec JAX-WS
- Approche Bottom / Up (à partir d'un POJO)
  - créer et annoter un POJO
  - compiler, déployer et tester
  - le document WSDL est automatiquement généré
- Approche Top / Down (à partir d'un document WSDL)
  - génération des différentes classes Java (JAXB et squelette du service web) en utilisant l'outil wsimport
  - compléter le squelette de classe de l'implémentation
  - compiler, déployer et tester

Approche Bottom / Up

- L'approche Bottom/Up consiste à démarrer le développement à partir d'une classe Java (POJO)
- Ajouter les annotations sur cette classe
- Déployer l'application sur un serveur d'application (ou via directement depuis Java SE 6)
- Le document WSDL est généré automatiquement en respectant les valeurs par défaut

  URL du WSDL : http://monserveur/app/Service?WSDL
- Toutes les méthodes du POJO sont des opérations du service web
- La surcharge de méthodes n'est pas supportée

Génération du WSDL

- Utilisation d'une interface pour définir les paramètres du service web
- Classe qui fournit l'implémentation du service web
  - pas nécessaire d'indiquer l'implémentation puisqu'elle est précisée dans l'annotation (vérification à l'exécution)
- L'outil wsgen génère des artifacts (JAXB, WSDL) à partir de classes Java annotées via JAX-WS
- Exemples d'utilisation
  - wsgen -keep -wsdl
    - génère les classes Java annotées JAXB
    - génère le document WSDL

Approche Top / Down

- L'approche Top/Down consiste à démarrer le développement à partir d'un document WSDL
- Le document WSDL est accessible via une URL ou via un fichier physique
- Utilisation explicite de l'outil wsimport pour la génération du squelette du service web
  - génération des classes liées à JAXB
  - génération des interfaces WS (interface décrivant le PortType)
- Création d'un POJO annoté @WebService en précisant l'emplacement de l'interface du portType
- Déployer l'application sur un serveur d'application
- Le reste du processus de développement est identique à celui de l'approche Bottom/Up

Annotations

- JAX-WS repose sur l'utilisation massive d'annotations pour la configuration d'un service web
- Les principales annotations sont les suivantes
  - @WebService : POJO implémentant un service web
  - @WebMethod : Paramétrer une opération
  - @WebParam : Paramétrer un message
  - @WebResult : Paramétrer un message de sortie
  - @WebFault : Paramétrer un message fault
- Noter que seule l'annotation @WebService est nécessaire
  - utilisation de valeurs par défaut

Annotations : @WebService

- Annote une classe Java pour définir l'implémentation du service web

- Attributs de l'annotation @WebService
  - String name : nom du service web
  - String endpointInterface : nom de l'interface décrivant le service web
  - String portName : nom du port
  - String serviceName : nom du service du service web
  - String targetNamespace : le namespace du service web
  - String wsdlLocation : l'emplacement du WSDL décrivant le Service

Annotation @WebMethod

- Annote une méthode d'une classe Java exposée comme une opération du service web

- Attributs de l'annotation @WebMethod
  - String action : l'action de l'opération. Dans le cas d'un binding SOAP, cela détermine la valeur de l'action SOAP
  - boolean exclude : précise que la méthode ne doit pas être exposée comme une opération. Ne pas utiliser dans une interface Java
  - String operationName : précise le nom de l'attribut name défini dans l'élément operation du document WSDL

Déploiement

- Un service web est déployé dans une application web (un service web par application web)

- Différentes catégories de serveur d'application pour gérer les services web avec JAX-WS
  - conteneur respectant la spécification JSR 109 (Implementing Entreprise Web Services)
    - la gestion du service web est transparente et maintenue par le serveur d'application
    - exemple : Glassfish
  - conteneur nécessitant une gestion par Servlet
    - nécessite une configuration explicite du service web
    - exemple : Tomcat

Développement client Java

- Le développement du client consiste à appeler des opérations du service web à partir d'un programme Java
- Le client peut être une application développée
  - Java SE (Swing, Eclipse RCP)
  - Java EE avec les EJB (JSP, Servlet, …)
- Possibilité de générer des appels aux Services Web de manière synchrone et asynchrone
- Le développeur ne manipule que du code Java, le code XML est caché (JAXB)

Développement client Java

- Le développement du client suit une procédure similaire à l'approche Top/Down où le point de départ est le document WSDL (via URL ou via fichier physique)
- Utilisation explicite de l'outil wsimport pour la génération du squelette du service web
- Génération des classes liées à JAXB
- Génération de classes service web (PortType et Service)
- Création d'une instance de la classe Service
- Récupération d'un port via get<ServiceName>Port()
- Invocation des opérations

Handlers

- Ce sont des intercepteurs permettant de réaliser des traitements lors de la réception et l'émission de messages
  - lors de la réception ils sont déclenchés avant l'appel à une opération
  - lors de l'émission ils sont déclenchés après l'appel à une opération
- Un handler est disponible dans la couche JAX-WS et par conséquent autant sur la partie cliente que celle du serveur
- Quand l'utiliser ?
  - pour filtrer les appels aux opérations d'un service web
  - pour l'écriture des logs
- Deux types de handlers
  - handlers liés au protocole de transport (ex : SOAP)
  - handlers liés au contenu transféré appelés logical handlers qui sont indépendants du protocole

Interface Handler

- Principales méthodes que l'on peut implémenter
  - boolean handleMessage(C context) : invoquée lors des messages entrants et sortants. Si false est retourné, le processus est arrêté
  - boolean handleFault(C context) : invoquée lors des messages en erreur (fault)
- Le type générique C hérite de MessageContext qui est une Map contenant un ensemble de propriétés
  - MESSAGE_OUTBOUND_PROPERTY : pour savoir s'il s'agit de messages entrants ou sortants
  - HTTP_REQUEST_HEADERS : pour récupérer l'en-tête HTTP de la requête
  - WSDL_OPERATION : nom de l'opération WSDL
  - WSDL_SERVICE : nom du service WSDL

Service Web avec les EJB

- A partir d'un EJB Session, il est possible de définir le POJO associé comme étant un service web
- Pour rappel, l'appel à un EJB Session passe obligatoirement par un client écrit en Java
- Les avantages à transformer un EJB Session en service web
  - caractéristiques des EJBs (transactions, sécurité, scalabilité, …)
  - plus grande hétérogénéité, client pas forcément écrit en Java
  - réutilisabilité du code
  - modularité (avec les annotations JAX-WS possibilité de masquer les méthodes qui ne doivent pas être découvertes)
- Nécessite d'avoir un conteneur EJB pour le déploiement

Service Web/EJB : serveur

- Partir d'une classe Java définissant un EJB Session (stateful ou stateless)
- Ajouter l'annotation @WebService
- Déployer l'application sur un serveur d'application ayant le support EJB (Glassfish par exemple)
- Le conteneur EJB s'occupe de la gestion du service web, aucune servlet n'est nécessaire
- Le document WSDL est généré automatiquement en respectant les valeurs par défauts

  URL du WSDL : http://monserveur/app/Service?WSDL
- Toutes les méthodes de l'EJB sont par défaut des opérations du service web

Service Web/EJB : client

- Le développement du client est similaire au client développé précédemment où le point de départ est le document WSDL (via URL ou via fichier physique)
- Utilisation explicite de l'outil wsimport pour la génération du squelette du service web
  - génération des classes liées à JAXB
  - génération de classes service web (PortType et Service)
- Injecter un @WebServiceReference pour récupérer une instance du Service à partir du conteneur EJB
- Récupération d'un port via get<ServiceName>Port()
- Invocation des opérations